Master 1 Informatique et Ingénierie des Systèmes Complexes (IISC)

Université de Cergy-Pontoise

Atelier de gesion de projet

Rapport de projet : COO

Auteurs :
Djahid ABDELMOUMENE
Amine AGRANE
Ishak AYAD
Abdelhakim SAID
Donald LAY
Yacine ZABAT

Sujet proposé par :
Mr. Tianxiao LIU
Mr. Dan VODISLAV

Rapport remis le 23 janvier 2020

Table des matières

Table des figures

1 Introduction à la partie C.O.O.

Dans le cadre de l'atelier de gestion de projet nous devions concevoir et implémenter un système permettant la création automatique de séjour en fonctions de critères donnés. Ce rapport traite de la conception des couches présentation, MVC contrôler, et métier (buisness). La couche persistence quant à elle est traitée dans un second rapport (rapport BDA).

2 Fonctionnalités du système

Notre système propose deux fonctionnalités principales :

— La première fonctionnalité consiste dans la recherche simple d'informations sur la base de données. On peut effectuer une recherche sur les hôtels et sur les sites touristiques. En ce qui concerne les hôtels, on effectue une recherche en fonction du nombre d'étoiles de l'hôtel. Dans la cas d'une recherche sur les sites touristiques, on spécifie le type du site (activité ou monument historique), et on a la possibilité d'ajouter des mots-clés dans les critères de recherche.

— La deuxième fonctionnalité est la construction automatique d'offres de séjours en fonctions de critères. Les critères sont le nom de l'hôtel, le budget pour le séjour, la durée du séjour, et la densité de visite des sites touristiques. Le système retournera les offres les plus cohérentes avec les critères donnés.

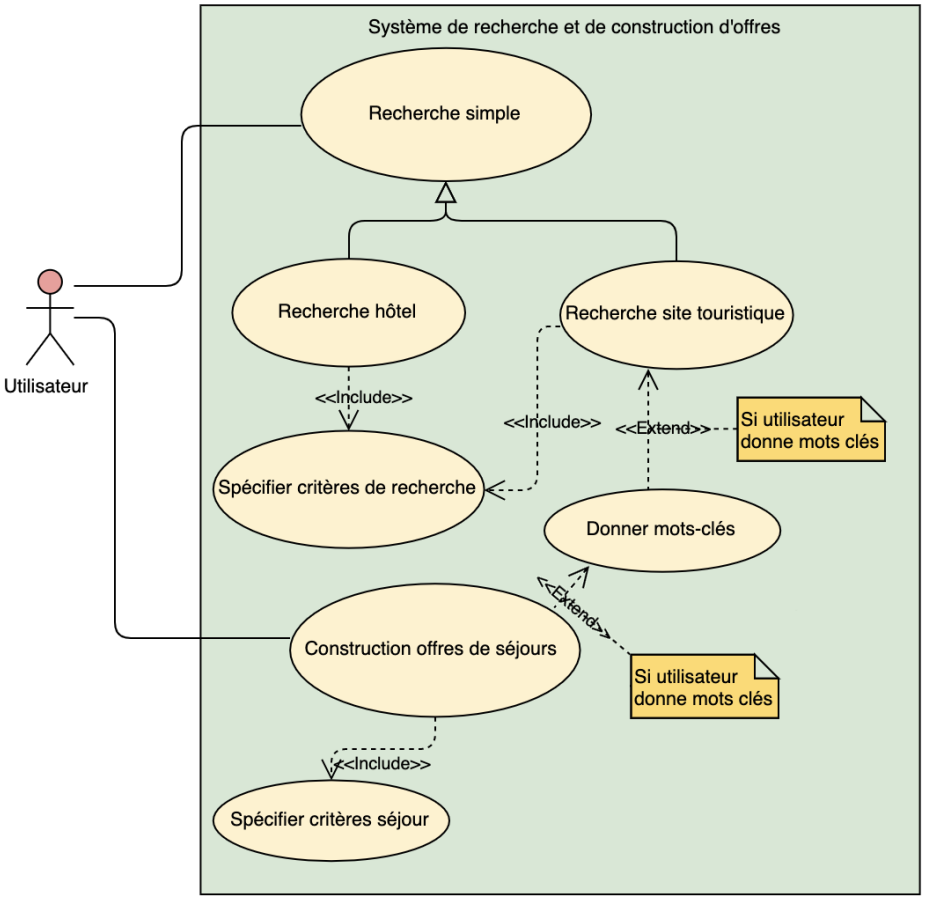

FIGURE 1 – Diagramme de cas d'utilisation du système

3 Conception du système

Notre application est composé des packages suivants :

— **beans** : contient les méthodes utilisées pour assurer les fonctionnalités sur la vue JSF ;
— **business** : contient les classes métiers assurant les différentes fonctionnalités du système. Ente autres, il contient notamment le sous-package engine utilisé pour la proposition d'offre d'excursions intelligente ;
— **exeception** : contient les exceptions pour les classes métiers de business ;
— **spring** : assure l'inversion de contrôle pour l'objet TripSimulation, LuceneBuilder, LucenePersistence et Persistence ;
— **bde** : implémente les fonctionnalités permettant l'exploitation de la base de données étendue (relationnelle et textuelle) ;
— **test** : contient les tests unitaires.

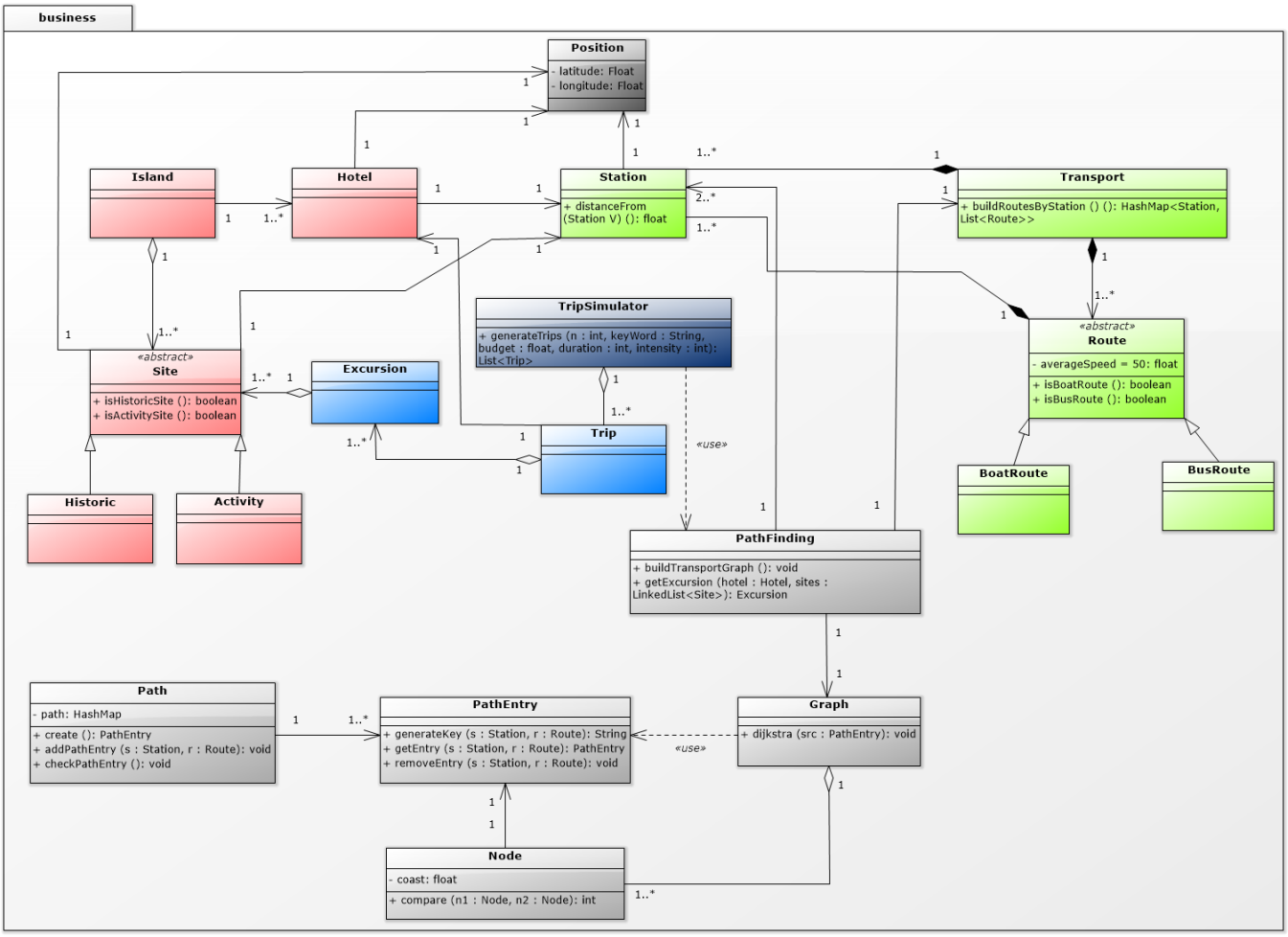

FIGURE 2 – Diagramme de classes du package buisness

4 Architecture de l'application

Notre application implémente une architecture 4-tiers qui assure la séparation des composants ayant des rôles différents. Cela garantit la maintenabilité et la portabilité de notre application.

4.1 Couche présentation

C'est la couche la plus haute au sein de notre application, elle se charge de la gestion de nos pages Web. Elle est implémentée par le Framework JSF Java qui assure la transmission des données et actions utilisateur vers les couches plus basses. Dans notre projet, on utilise les fichiers JSF suivants :

- navbar.xhtml : Un fichier jsf utilisé pour modéliser notre barre de navigation. Ce fichier est inclus dans tous les autres fichiers JSF.
- home.xhtml : Fichier JSF pour la page d'accueil.
- search.xhtml : Fichier JSF pour la page des recherches simples, les valeurs des champs sont sauvegardés dans les attributs d'une classe de la couche suivante (SearchBean), ce fichier gère aussi l'exécution des méthodes de ce bean.
- trips.xhtml : Fichier JSF pour la page de construction des offres.

4.2 Couche contrôleur MVC

Nos contrôleurs sont implémentés par des classes de type "Java bean", ces contrôleurs gèrent le routage dans notre application Web et constituent une couche intermédiaire pour raccorder la partie UI avec le coeur du système (parti business)

NavigartonBarBean : ce contrôleur est appelé par la vue de notre barre de navigation, il sert comme un objet de routage. À chaque fois qu'une méthode est appelée, il renvoie le nom de la vue suivante.

SearchBean et Trips Bean : Ces contrôleurs sauvegardent les attributs saisies par l'utilisateur et fournies par la vue "Search JSF" ou "Trips JSF" pour les repasser et appeler les bonnes méthodes de la couche métier afin de récupérer les bons résultats.

4.3 Couche métier

Cette couche se charge de la gestion de la logique de notre système, à l'aide de la classe TripSimulation, elle manipule un nombre important de classes se trouvant dans les couches au-dessus, et elle assure l'exécution des requêtes à partir des bon opérateurs, (SQL et Lucene). Cette classe utilise aussi des classes du système pour répondre à certaines opérations, par exemple l'appel de la classe Pathfinding pour faire des calculs et construire des offres.

5 Injection par Spring

On constate qu'il existe un certain nombre de dépendances entre les différents composants de notre système, chaque couche utilise des composants de la couche supérieure et/ou inférieur.

Afin de répondre à cette problématique, les méthodes traditionnelles (passage de dépendances dans les constructeurs ou initialisation de dépendances dans les constructeurs) ne garantissent pas la maintenabilité du code.

On utilise alors Spring afin de d'apporter une solution optimale, les dépendances de chaque classe seront définies dans le fichier spring.xml. Lors de la mise en marche du système, on récupère notre objet principal de métier (TripSimulation) à partir du Framwork spring qui va injecter toutes les dépendances à l'aide de nos définitions dans le fichier spring.xml.

6 Design patterns

Au sein de notre code, de nombreux design patterns ont été implémentés dans le but d'avoir une conception optimale.

— **Singleton** : La classe "JdbcConnection" est la classe chargée d'établir la connexion avec notre base de données. Nous n'avons besoin que d'une unique connexion à notre base de données, on utilise alors le pattern Singleton afin de restreindre l'instanciation de la classe à un seul objet.

— **Builder** : Le patron Builder est utilisé dans l'algorithme de constructions des séjours. Il est également utilisé dans la classe "LuceneBuilder", classe qui se charge de la création des fichiers de description dans le répertoire data.

— **Facade** Le design pattern Facade est employé au sein de l'api dans la classe "ExtendedDB". L'objectif était de cacher la complexité de conception et d'offrir une interface simple du système.

— **Template method** : On utilise le pattern Template method dans le package "transport". Ce package contient la classe abstraite "Route" dont les méthodes seront implémentées dans les classes "BoatRoute" et "BusRoute", ces méthodes servent à préciser quelle est le type de l'objet.

— **Iterator** : On utilise le pattern Iterator afin d'implémenter les opérateurs SQL, Lucene et la jointure qui sont des résultats de requêtes .

— **Flyweight** : On utilise le design pattern Flyweight dans la classe "PathEntry", afin de limiter l'instanciation d'un nombre important d'objets.

7 Tests unitaires

Afin de s'assurer de l'efficacité et de la cohérence de notre système, de nombreux tests ont été effectués. L'ensemble de nos tests manuels et automatisés se trouvent dans le package "tests" du projet.

7.1 Tests automatisés

Nous avons utilisé la librairie JUnit afin d'automatiser nos tests. Les test automatisé portent sur les algorithmes de "Pathfinding" et "Excursion" présents dans la couche Buisness. On utilise des mocks dans nos tests unitaires afin de palier à la problématique de dépendances des classes entre elles. Typiquement, lors du test de notre classe "PathFinding", on créera un ensemble d'objets afin de simuler les comportements des obejts réel qui seront manipulé par notre classe. Les objets sont simulés via les méthodes initStations() et initRoutes().

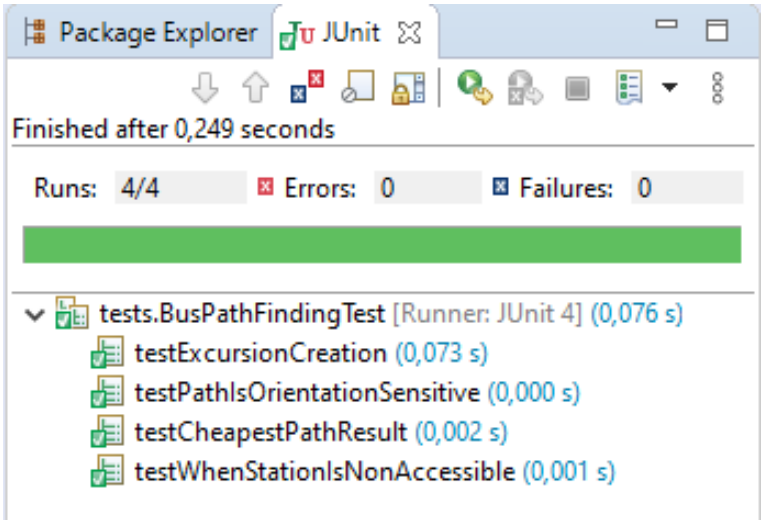

FIGURE 3 – Tests automatisés à l'aide de JUnit

7.2 Tests manuels

Les tests manuels concernent principalement les fonctionnalités mettant en oeuvre le moteur d'indexation Lucene :

— **LuceneTester** : Test l'indexation de fichiers se trouvant dans un répertoire et la recherche textuelle sur cet index ;

```
Indexing C:\Users\ASUS\eclipse-workspace2\AGP\src\luceneSRC\data\8.txt
Indexing C:\Users\ASUS\eclipse-workspace2\AGP\src\luceneSRC\data\9.txt
63 File indexed, time taken: 63 ms
id: 22,score: 1.617209
id: 22,score: 1.617209
id: 58,score: 1.3537517
id: 58,score: 1.3537517
id: 59,score: 1.3465924
id: 59,score: 1.3465924
id: 62,score: 1.260104
id: 62,score: 1.260104
id: 16,score: 1.2263906
id: 16,score: 1.2263906
```

FIGURE 4 – Résultat du test manuel LuceneTester

— **LuceneExempleTest** : Même test que LuceneTester mais en n'indexant qu'un seul fichier ;
— **ExtendedDBTest** : Test de l'API permettant l'exploitation des données de la base de données étendue requête SQL, textuelle et mixte (en suivant les deux plans d'exécution décrits dans le sujet).

8 Algorithme de construction des offres

L'algorithme de construction de l'offre se compose de deux parties, la première génère des paires d'hôtels et des listes de sites à visiter, et la seconde les utilise pour générer des excursions pour le voyage.

8.1 Recherche des sites

Dans cet algorithme, nous commençons par sélectionner les hôtels en fonction du niveau de confort et du budget préférés de l'utilisateur. Pour ce faire, nous avons pour chaque niveau de confort un pourcentage prédéfini pour la tranche du budget total de l'hôtel. Nous trouvons ensuite les meilleurs hôtels dont le prix est le plus proche de ce pourcentage.

Ensuite, nous avons une liste d'hôtels, dont chacun doit correspondre à une offre, nous utilisons ensuite chaque hôtel pour générer une liste de sites à visiter pour l'offre correspondante. Pour cela, nous utilisons l'API SQL étendue pour obtenir les niveaux de pertinence de chaque site pour les mots clés donnés par l'utilisateur, nous trions ensuite les sites en fonction d'une heuristique qui utilise à la fois la pertinence et la distance par rapport à l'hôtel sélectionné.

Nous utilisons ensuite la liste triée des sites et obtenons autant de sites (tels qu'ils apparaissent dans la liste) que le budget le permet.

8.2 Génération des excursions

Pour générer des excursions pour le voyage, nous avons d'abord besoin d'un algorithme qui trouve le chemin le moins cher pour aller d'une station à l'autre.

8.2.1 Recherche de chemin optimale

Pour rechercher le chemin le moins cher, nous utilisons l'algorithme de Dijkstra avec un graphe orienté qui représente l'ensemble du réseau de transport, nous utilisons toutes les lignes de transport qu'elles soient un bus ou un bateau.

La construction du graphe se fait comme suit : nous faisons d'abord de toutes les différentes paires station/route des sommets du graphe, nous connectons ensuite tous les sommets consécutifs dans une route avec un arête de coût 0, ceci reflète le fait qu'une fois que nous sommes dans un bus/bateau nous n'avons rien à payer. Ensuite et pour chaque station nous prenons toutes les routes qui passent par celle-ci et nous les connectons entre elles avec un arête avec une valeur correspondant au prix du ticket de route du sommet de destination, ceci correspond au fait que nous devons payer pour un nouveau ticket si nous décidons de changer de ligne de transport.

Comme dernière étape de la construction du graphe, nous ajoutons deux nœuds nommés "S" et "E", qui signifie Start et End, et nous connectons le nœud S avec les sommets qui correspondent à la station de départ et nous connectons tous les sommets pour la station de destination souhaitée au sommet E. Ceci marque la fin de la construction du graphe qui ne doit être faite qu'une seule fois au début, en ne changeant que les liaisons aux nœuds S et E selon les besoins.

Pour trouver le chemin le moins cher, nous appliquons simplement l'algorithme de Dijkstra sur le graphe résultant et nous retrouvons le chemin entre les sommets S et E.

8.2.2 Géneration des excursions

Pour générer les excursions, nous utilisons une forme d'algorithme de Nearest Neighbor pour regrouper des clusters de sites, de plus nous limitons également le nombre de sites visités pendant une excursion en utilisant une notion approximative de durée.

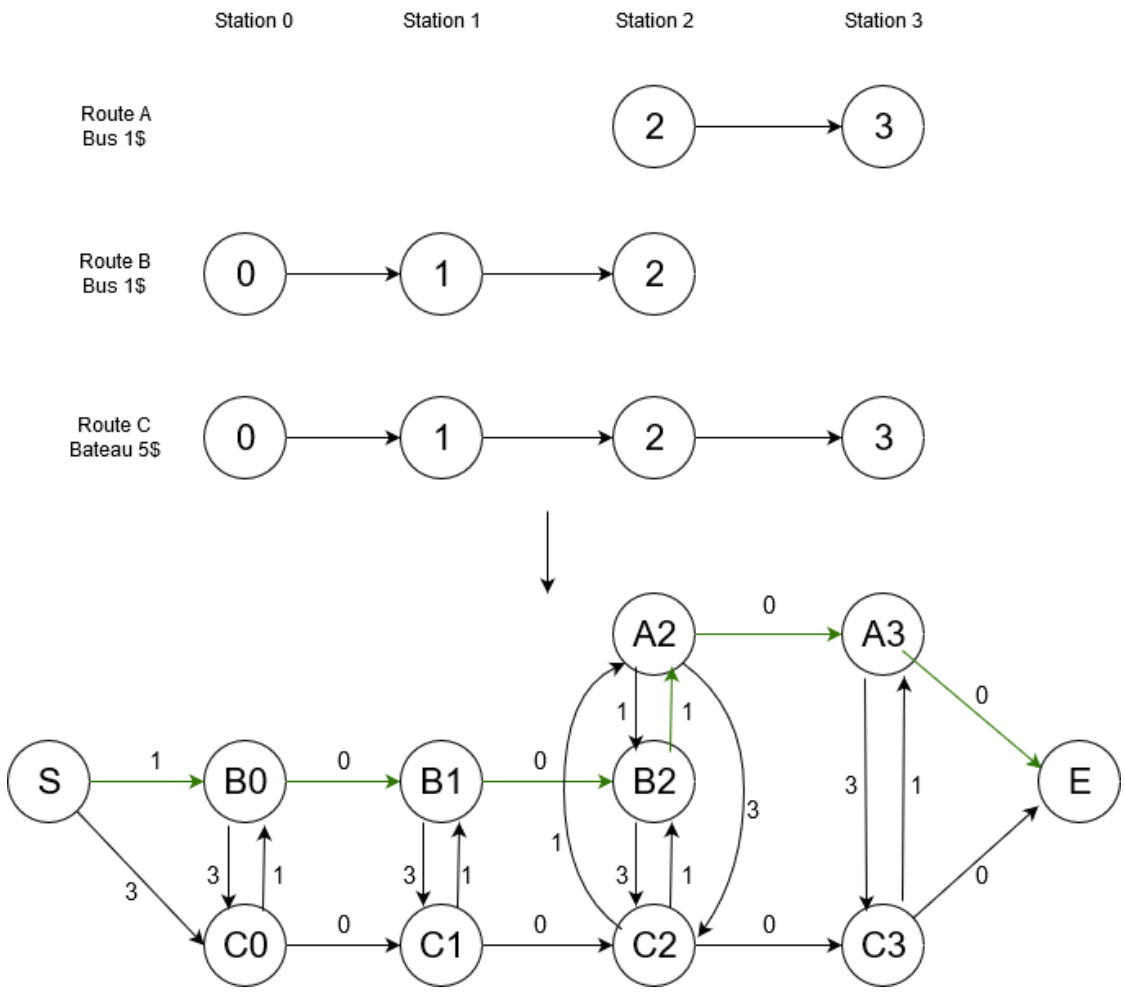

FIGURE 5 – Exemple de recherche de chemin avec un réseau de transport contenant 3 routes et 4 stations (entre station 0 et station 3)

La façon dont l'algorithme fonctionne est que nous commençons à l'hôtel, puis nous prenons un site (au hasard au début), et ensuite nous calculons le temps que ça nous prendrait de : 1) aller le visiter (en utilisant l'algorithme décrit précédemment) 2) passer du temps sur le site, et 3) retourner à l'hôtel. Nous comparons ensuite ce temps à une durée seuil que nous définissons pour chaque niveau de confort, et si la durée est inférieure à ce seuil, nous ajoutons ce site à l'excursion. Nous continuons à faire la même chose en déplaçant la station " actuelle " de l'hôtel à la station du site choisi.

Après avoir déplacé la station au nouveau site, nous pouvons utiliser l'algorithme du chemin le moins cher pour trouver le voisin le plus proche de la station actuelle (en fonction des prix de transport), et nous faisons la même comparaison de seuil comme avant, et nous passons à la station de ce nouveau site. On continue à faire cela jusqu'à ce qu'on ait épuisé le temps prédéfini dans notre seuil ou bien jusqu'à ce qu'on ait épuisé les sites. Nous gardons également une horloge tout au long de l'algorithme pour suivre l'heure actuelle de la journée, que nous mettons à jour chaque fois que nous rajoutons un nouveau site.

À la fin, lorsque nous épuisons le sites à visiter ou que nous dépassons notre seuil, nous marquons la fin d'une excursion et on l'ajoute à notre liste d'excursions pour le séjour. Et si nous avons encore d'autres sites à visiter, nous recommençons tout ce processus, jusqu'à ce qu'il

n'y reste plus de sites.

9 Remerciements

Notre groupe tiens à remercier Monsieur Tianxiao LIU et Monsieur Dan VODISLAV pour nous avoir confier ce projet. Au cours de cette semaine, nous avons eu l'opportunité d'approfondir nos connaissances et nos compétences en JAVA et en base de données. Sortir du cadre théorique et avoir l'occasion de travailler en équipe sur un projet d'une telle envergure nous a fait comprendre les responsabilités qu'impliquait la gestion de projet, et nous a fait gagner en maturité.